АРХИТЕКТУРНИ СТИЛОВЕ

Outline

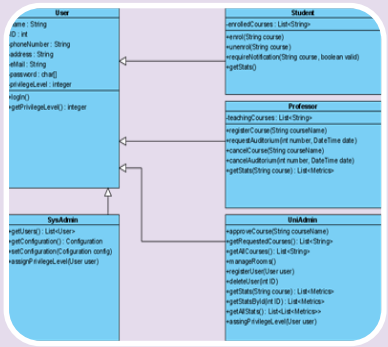
- Some reflection on software architecture
- Definition of architectural styles
- Different styles
  - Pipe-and-Filter
  - Layered
  - Client-server
  - Repository/Blackboard
  - Model-View-Controller
  - Implicit invocation/Message passing

Software architecture reflection

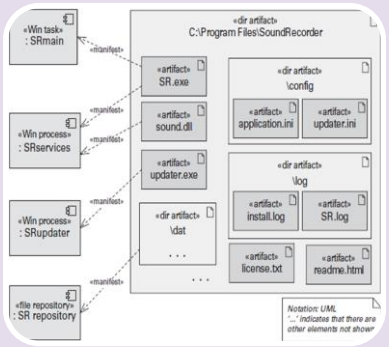
- What is software architecture?
  - A collection of structures, that represent different view points over the system
  - Each structure consists of elements, their externally visible characteristics and the connections between them
  - Structures are represented by views

4+1 Software architecture views

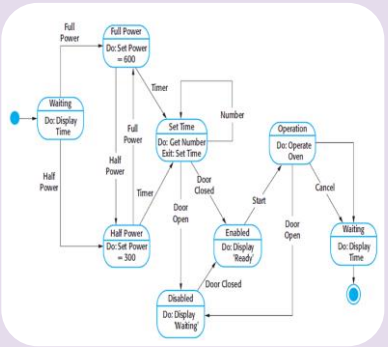
(by Kruchten)



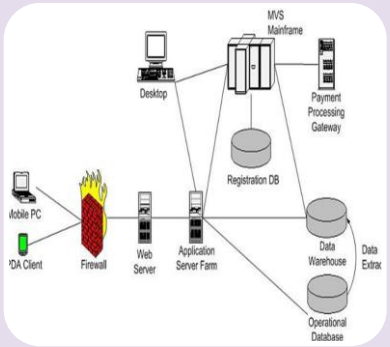
Logical view



Code view



Process view



Deploy ment view

Sources: http://www.agilemodeling.com

Software Engineering by Ian Sommerville, 9th edition (2010), Addison-Wesley Pub Co;

Architectural styles

- Logical view of software has four levels of abstraction
  - Components and connectors
  - Their interfaces
  - Architectural configurations – specific topology of interconnected components and connectors
  - Architectural styles – patterns for successful and practically proven architectural configurations

More precisely

- Architectural style defines a family of systems in terms of a pattern of structural organization.
- Styles determine:
  - The vocabulary of components and connectors that can be used in instances of that style
  - A set of constraints on how they can be combined, like:
    - The topology of the descriptions (e.g., no cycles)
    - Execution semantics (e.g., processes execute in parallel)

What is a component

- Computational unit that have a particular functionality, which is accessible via well defined interfaces
  - Input interface, which specifies what the component require in order to execute its functionality
  - Output interface, which specifies what the component claims to do, given that everything from input interface specification was fulfiled

What is a connector

- First class entity, which represent the communication mechanism (the protocol) between components
  - Connectors also have interfaces, sometimes called roles
- Both components and connectors may be reusable

Architectural styles

- Pipe-and-Filter
- Layered
- Client-server
- Repository/Blackboard
- Model-View-Controller
- Implicit invocation/Message passing
- Others (next week)

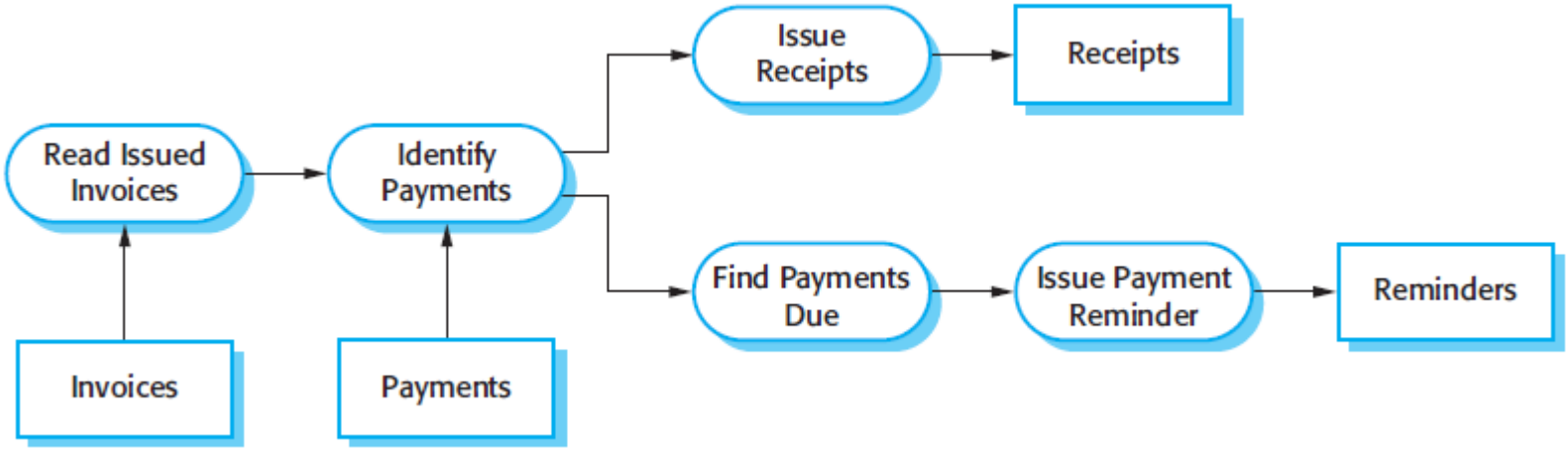
Pipe-and-Filter style

- Each component (filter) in the system transfers the data in consecutive order to the next component
- The connectors (pipes) between filters represent the actual data transfer mechanisms

Pipe-and-Filter style

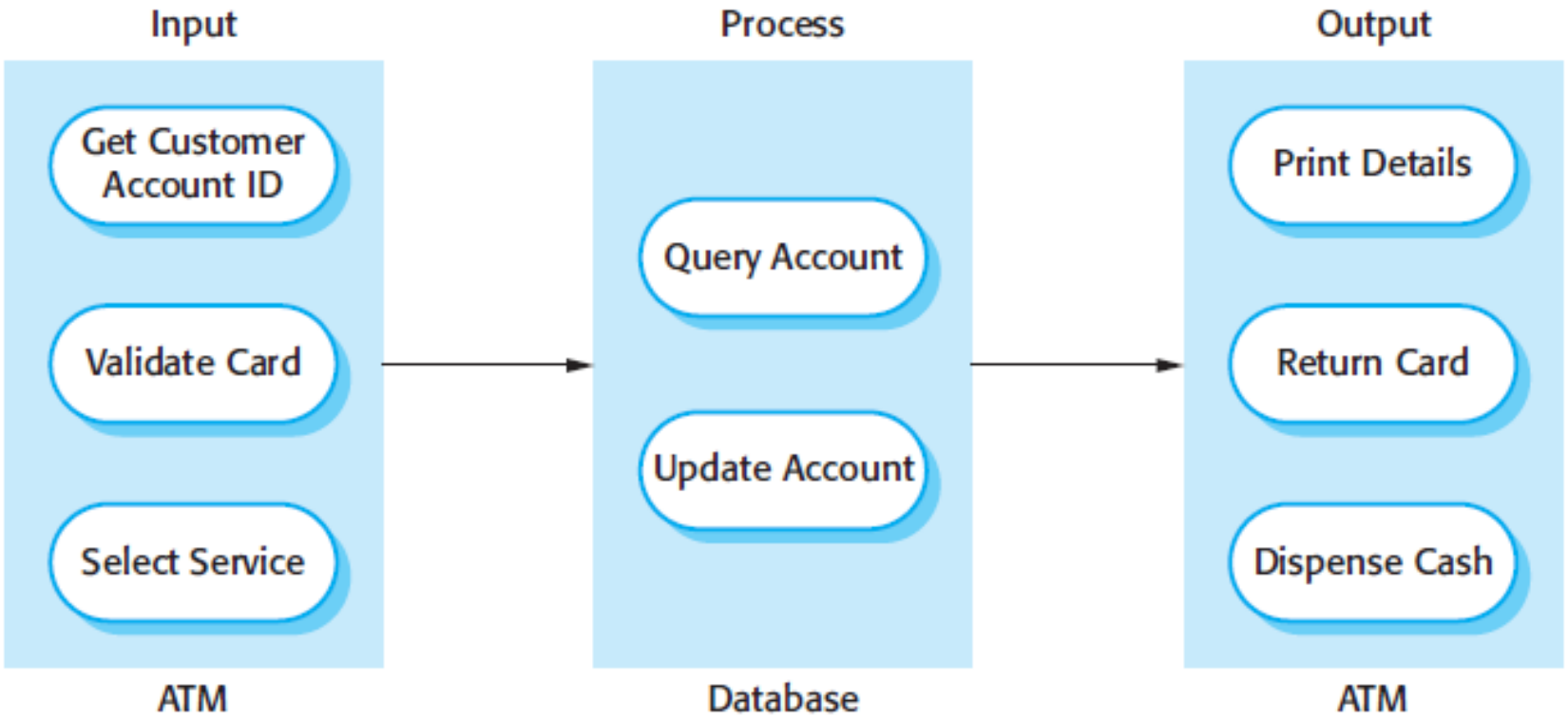
- The name ‘pipe and filter’ comes from the original Unix system where it was possible to link processes using ‘pipes’.
  - Pipes passed a text stream from one process to another.
- Filters represent ***computational units*** in the system.
  - In other words the functionality is there.
  - They read data via their input interfaces, then process it and finally send the data to their output interfaces
  - Filters don’t have information about their neighbors
- Pipes have the duty to transfer the data from the output of a filter to the input of the next filter

Pipe-and-Filter style – examples



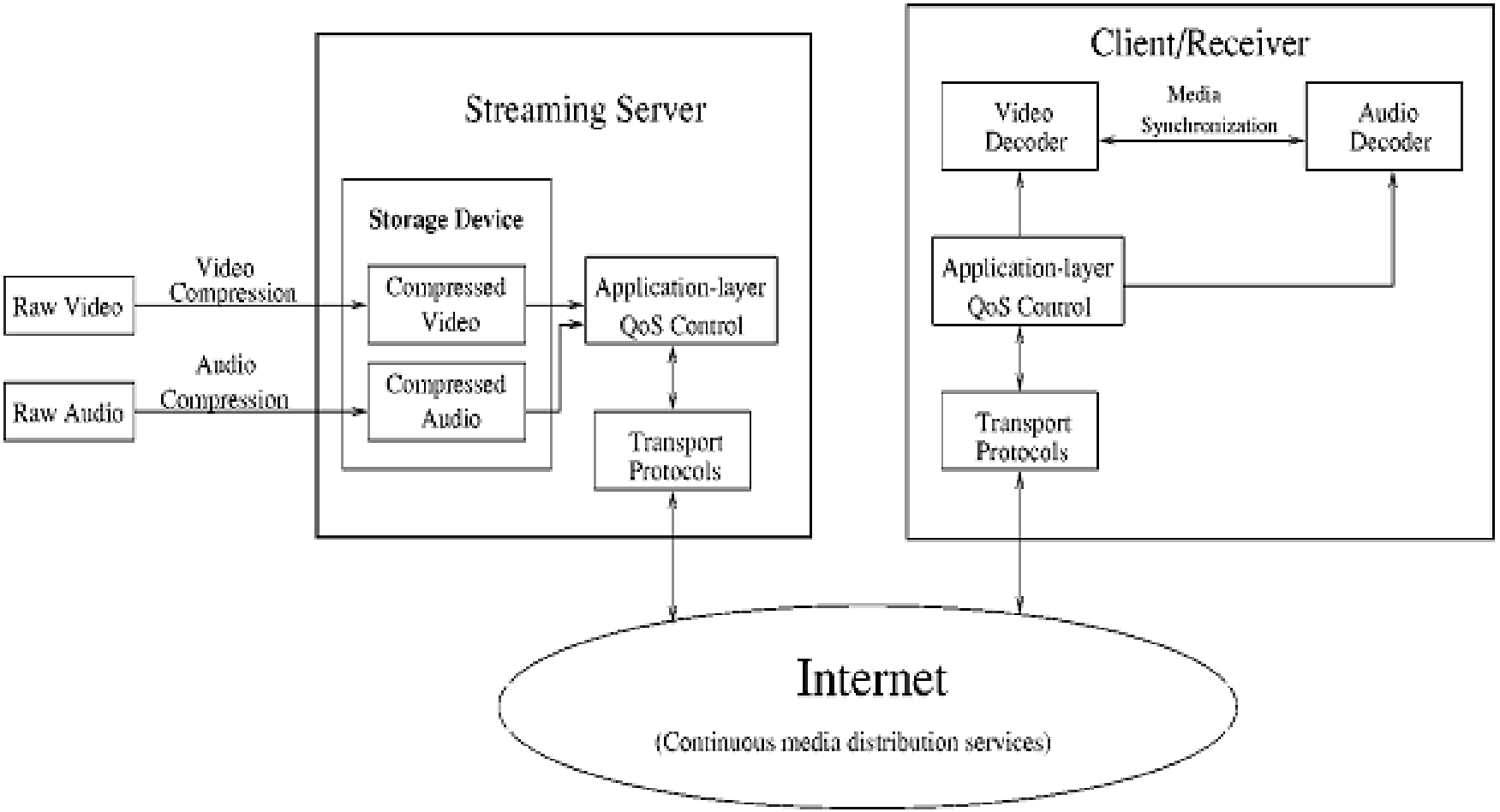
Source: Software Engineering by Ian Sommerville, 9th edition (2010), Addison-Wesley Pub Co;

Software architecture of an ATM system



Source: Software Engineering by Ian Sommerville, 9th edition (2010), Addison-Wesley Pub Co;

Video streaming architecture



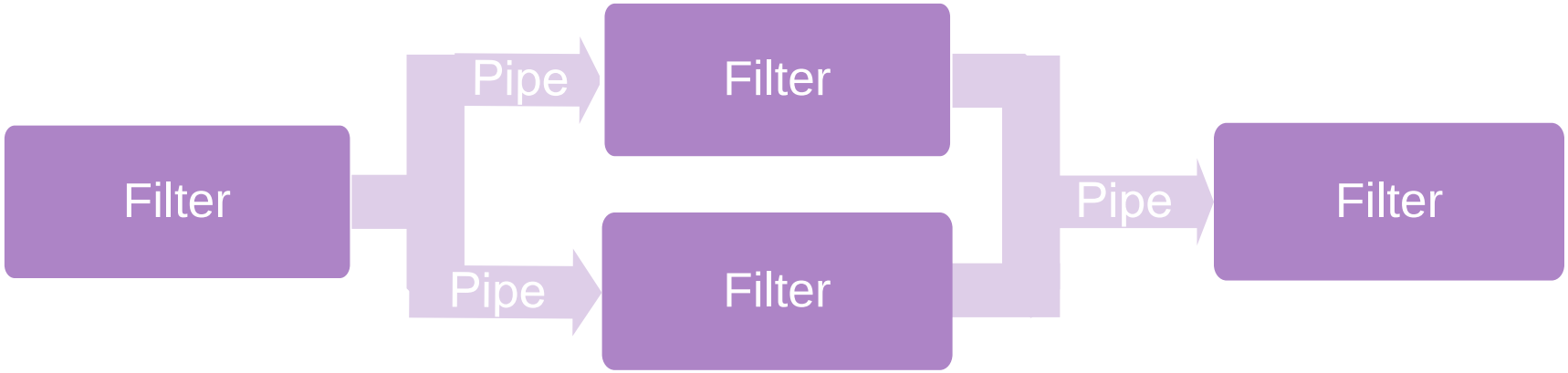
Source: Streaming Video over the Internet: Approaches and Directions, Dapeng Wu, Yiwei Thomas Hou et. al.

What alternatives of this style we may have?

Variations of pipe-and-filter style

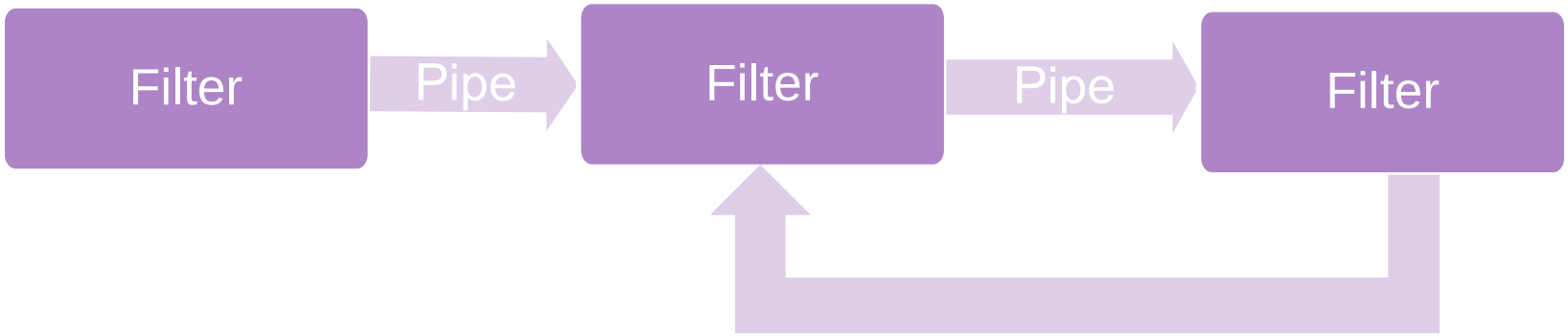
- Batch-sequential

- Parallelism/redundancy



Variations of pipe-and-filter style

- Loopback



Variations of pipe-and-filter style

- In terms of communication protocol
  - **Pipeline/stream** – Data processing may start immediately after the first byte is received by the filter
  - Compare with **batch-sequential style** that requires the whole data to be transferred before the filter starts working with it

Advantages of pipe-and-filter style

- Intuitive and easy to understand
- Filters stand alone and can be treated as black boxes, which leads to flexibility in terms of maintenance and reuse
- Easy to implement concurrency (not in batch-sequential)
- It is straightforward applicable to the structures of many business processes
  - Easy to use, when the processing required by an application can easily be decomposed into a set of discrete, independent steps

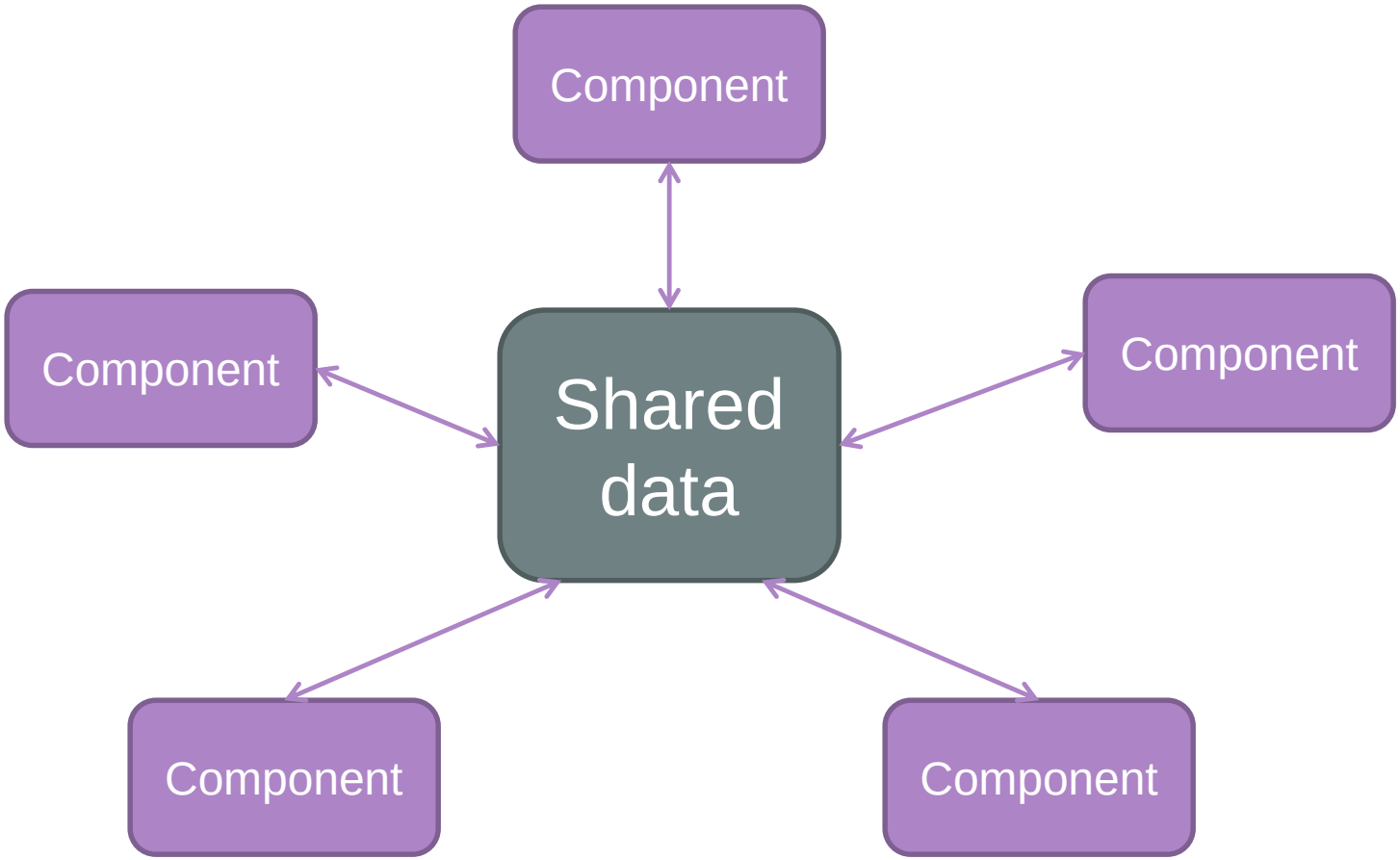
Disadvantages of pipe-and-filter style

- Due to the sequential steps of execution, it is difficult to implement interactive applications.
- Poor performance
  - Each filter has to parse/unparse data
  - Difficult to share global data
- Filters must agree on tor data format

Pipe-and-Filter style - some issues

- **Complexity –** In distributed environment if the filters are executing on different servers
- **Reliability –** Use an infrastructure that ensures data flowing between filters in a pipeline will not be lost.
- **Idempotency** – Detection and removal of duplicate messages
- **Context and state** – Each filter must be provided with sufficient context with which it can perform its work, which may require a considerable amount of state information.

Shared-data style



Shared-data style

- Actively used in systems, where components should transfer large amounts of data
- The shared-data may be seen as a connector between the components
- shared-data style – variations
  - ***Blackboard*** – when any data is send to the shared-data connector, all components should be informed about this. In other words the shared-data is an active agent
  - ***Repository*** – shared-data is passive, no notifications are send to the components

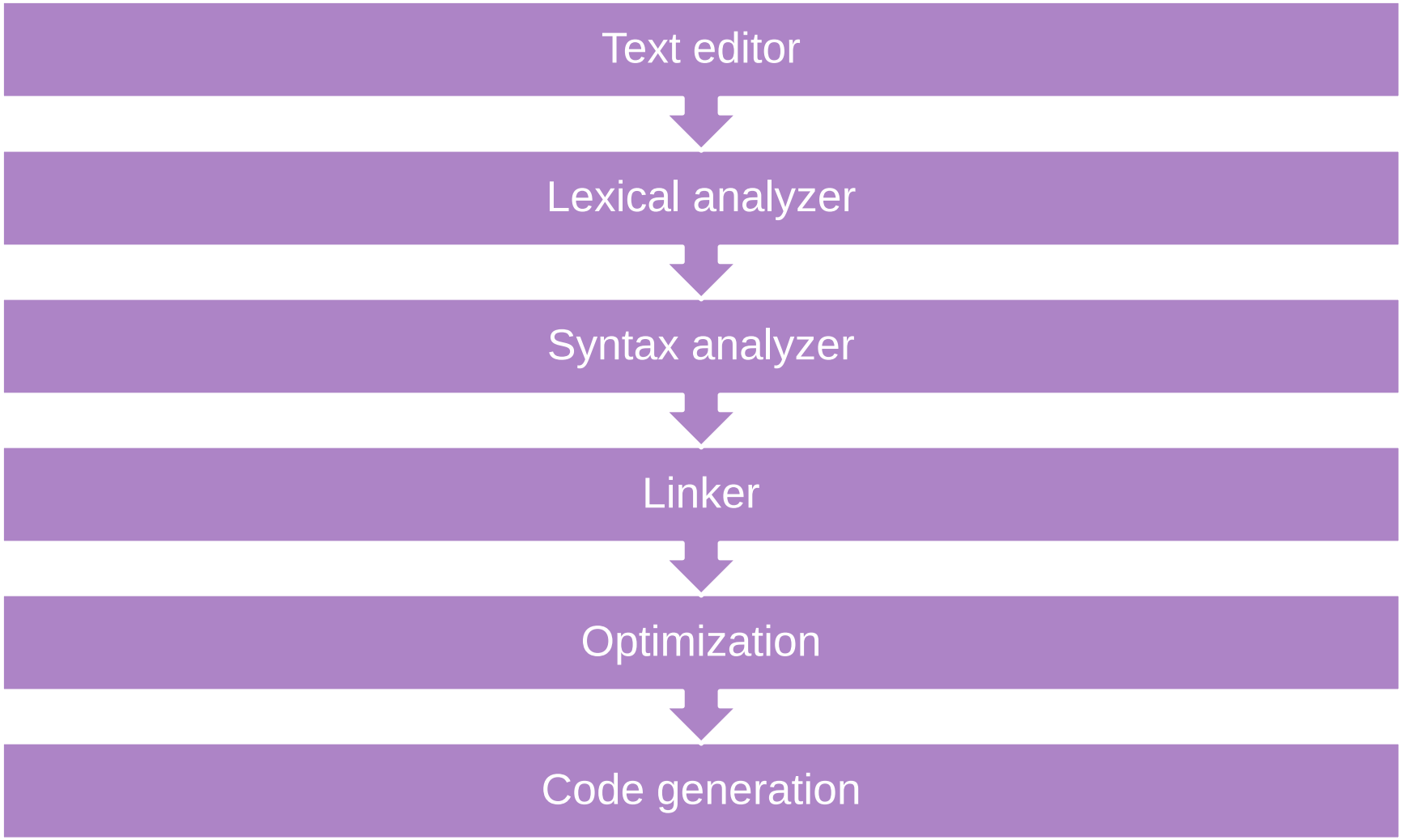
Advantages of shared-data style

- Scalability – knew components may be added
- Concurrency – all components may work in parallel
- Highly effective when large amounts of data are exchanged
- Centralized management of data
  - Better conditions for security, backup, etc.
  - The components are independent of the data producer

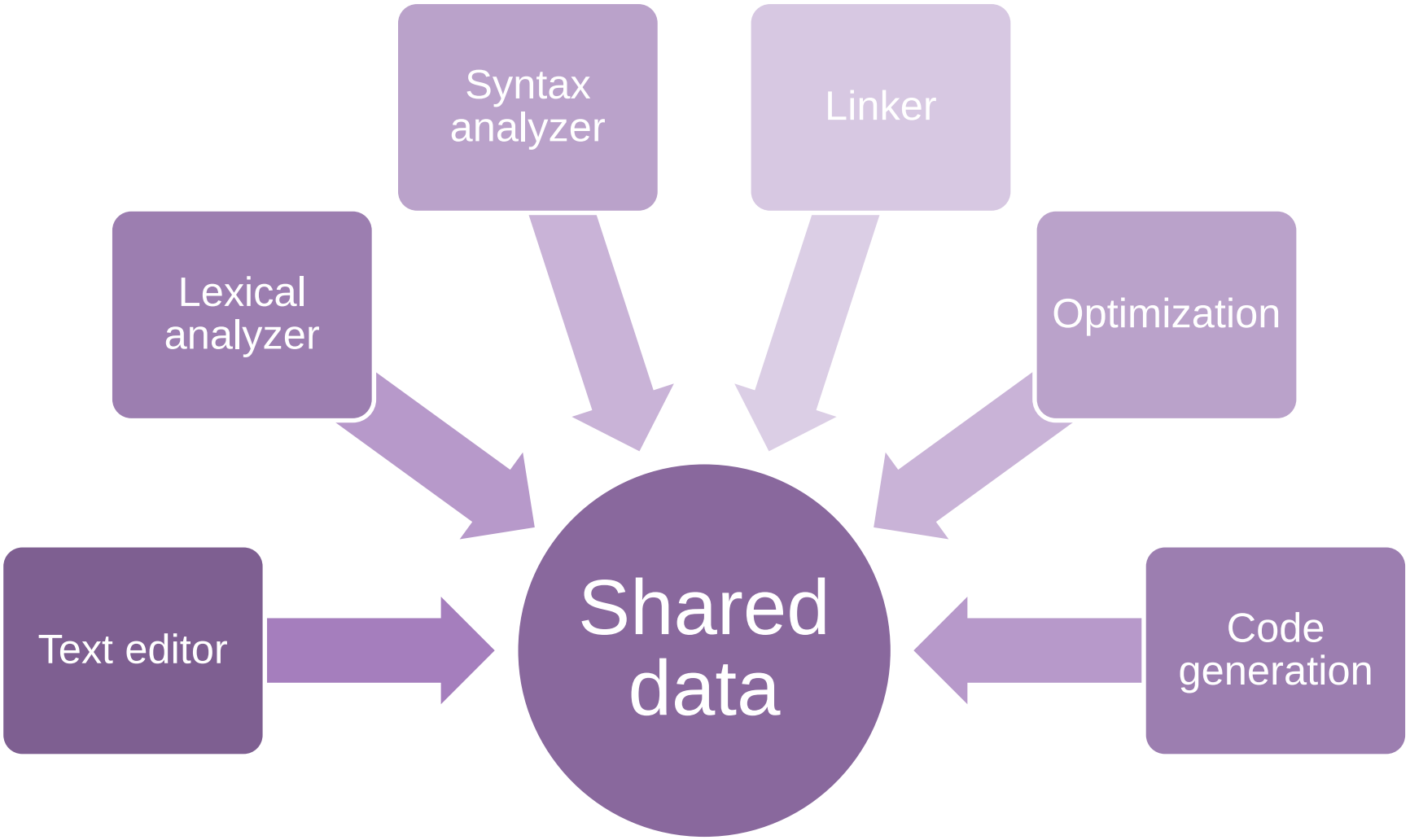
Disadvantages of shared-data

- Difficult to apply in distributed environment
- The shared-data should maintain uniform data model
  - Changes in the model may lead to unnecessary expenses
  - Tight dependency between the blackboard and the knowledge source
- It may become a bottleneck in case of too many clients

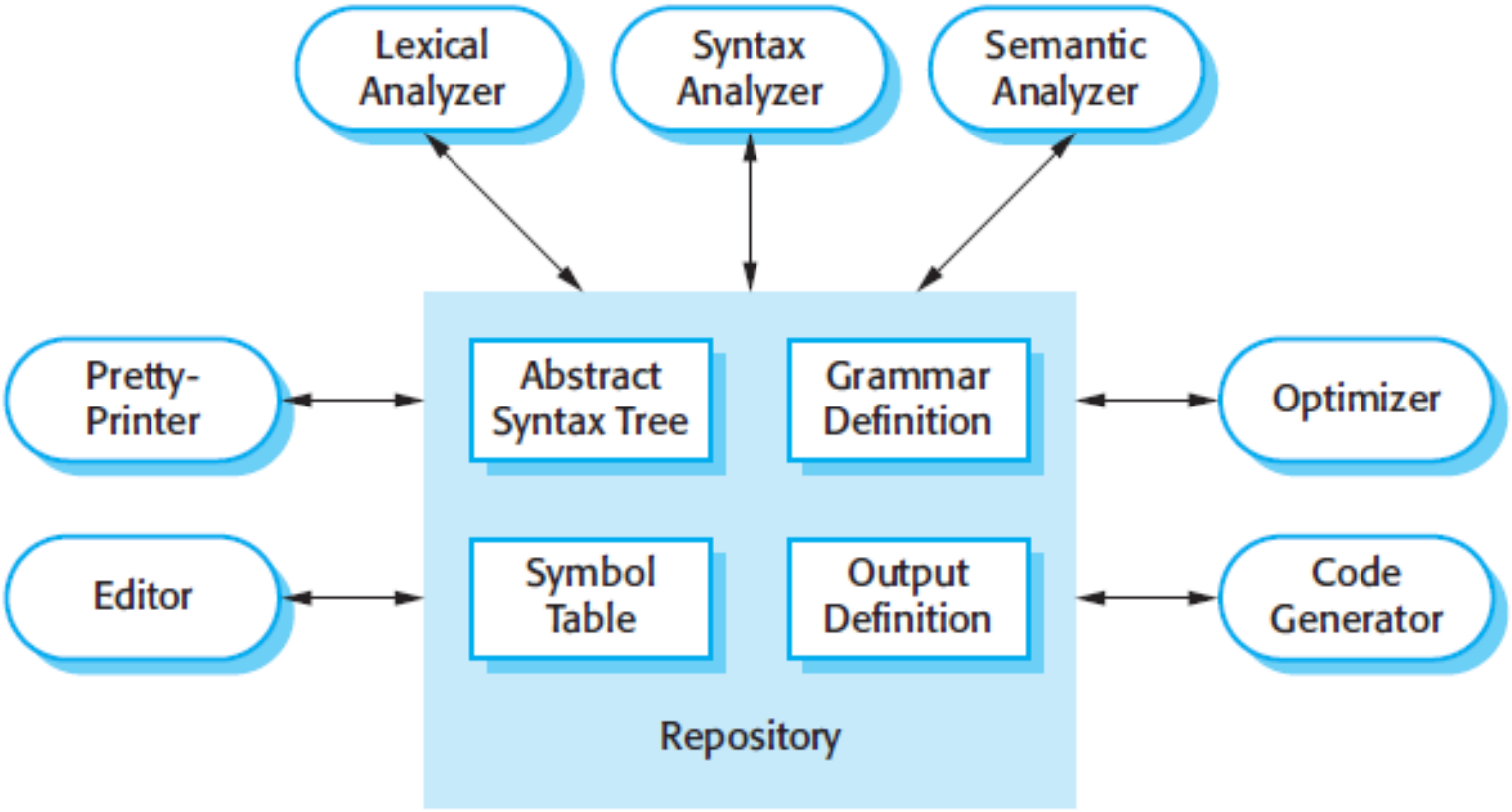
Compiler architecture (pipeline)



Compiler architecture (shared data)

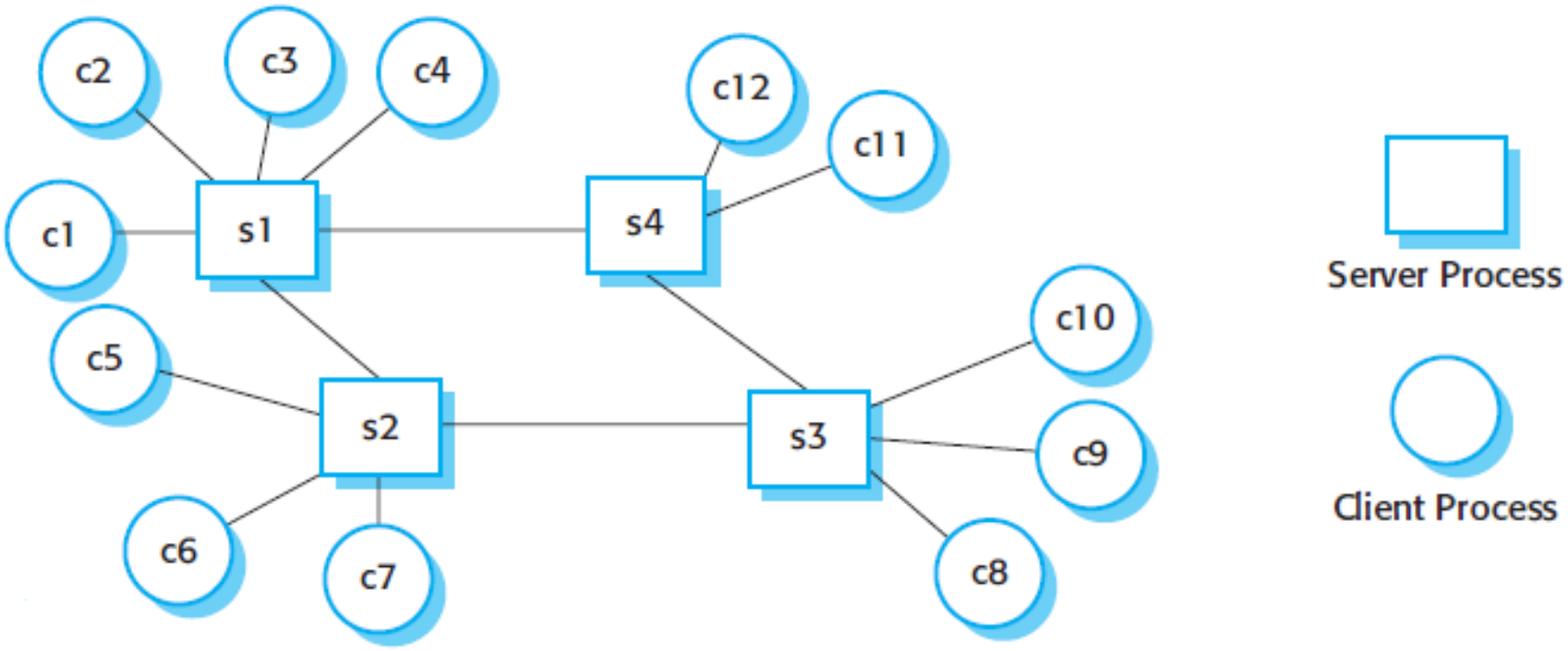


Compiler architecture (more details)



Source: Software Engineering by Ian Sommerville, 9th edition (2010), Addison-Wesley Pub Co;

Client server style

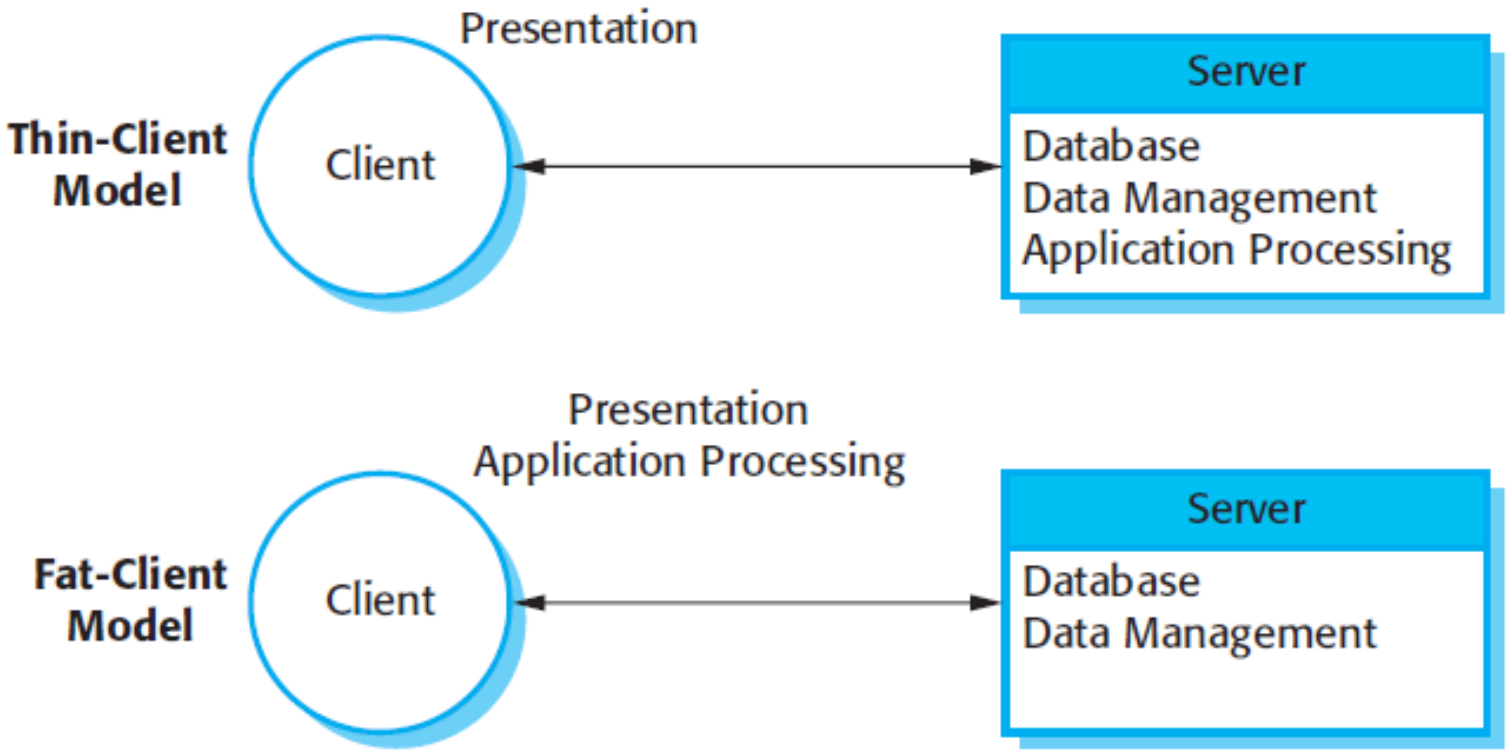


Source: Software Engineering by Ian Sommerville, 9th edition (2010), Addison-Wesley Pub Co;

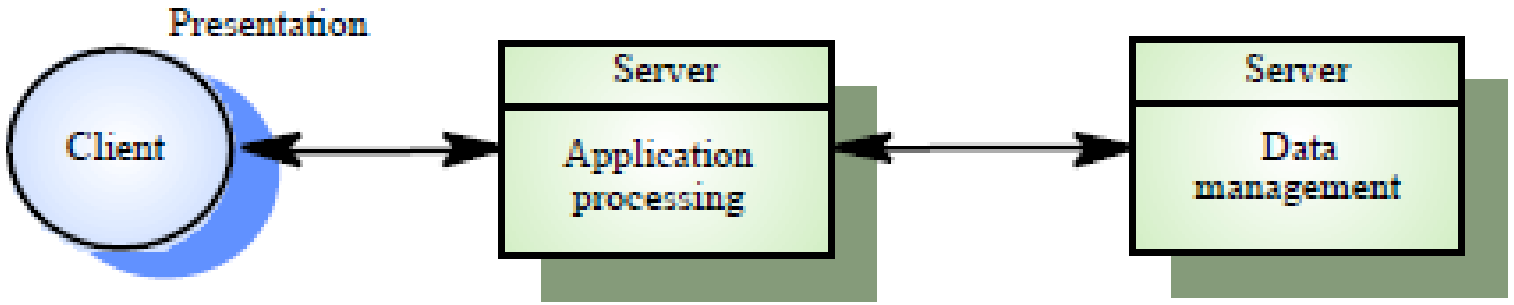
Client-server style

- System is designed as a set of servers that offer services and a number of clients that use these services
- Servers are not necessary to have information about his clients
- Classical implementation – *thin* client
  - The client implements user interface functionality
  - The server implements the data management and application processing functionality
- Fat clients may implement some of the application processing functionality

Thin or *Fat* client-server



Three tier client-server model



- Better performance
- Better security
- …

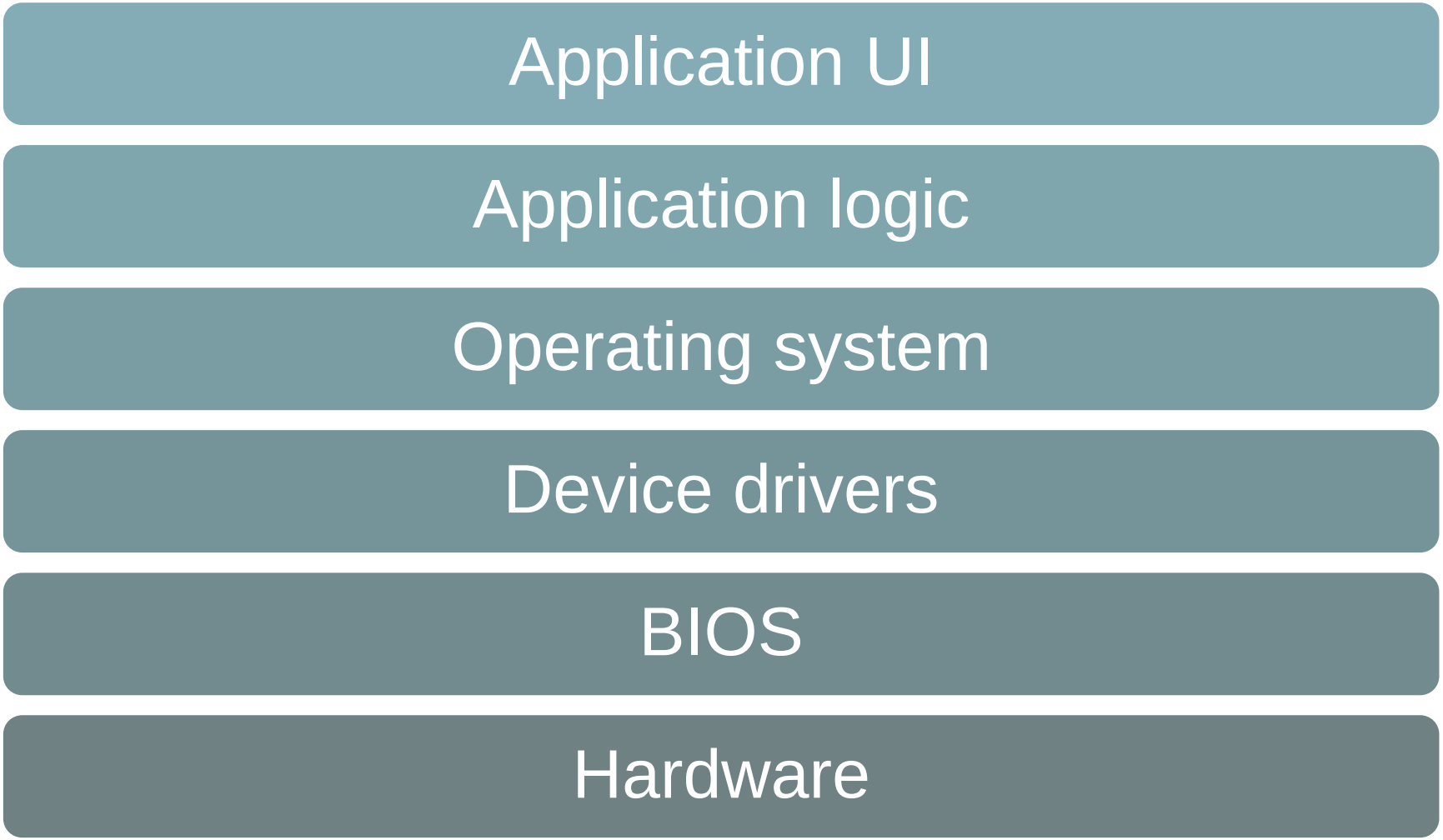
Advantages of client server style

- Centralization of data
- Security
- Easy to implement back-up and recovery

Disadvantages of client server style

- Server workload may be increased with large number of clients
- What will happen if the server fails
  - Needs redundancy/fault-tolerance

Layered style



Rules of the layered style

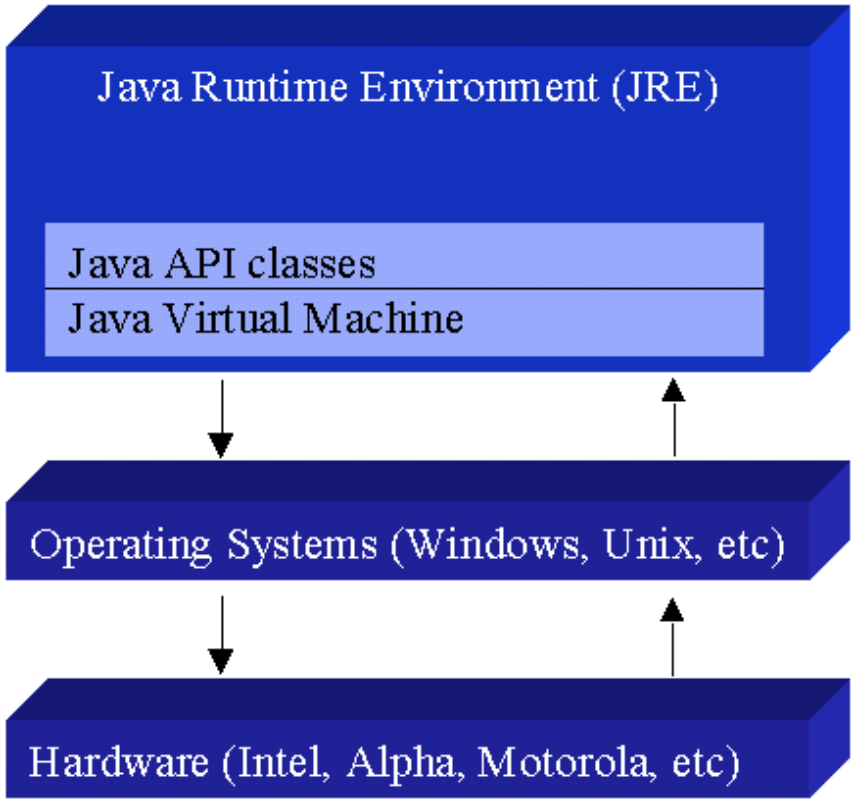
- Represent the system as organized by hierarchically ordered layers
- Classical implementation
  - Each layer offers services via an interface, but only for the layer which is directly above it and uses the services from the layer which is directly below it
  - This way a layer represents a
    - Server – for the layer above
    - Client – for the layer below
  - The interfaces may be similar to APIs (Application Programming Interfaces)

Layered style

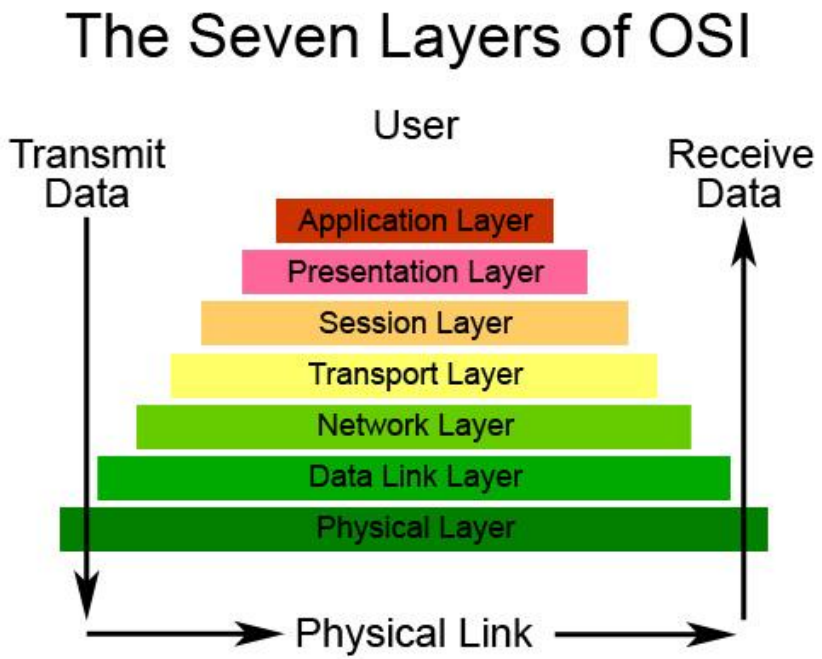
- Practically, this is the most widespread style in all kinds of software systems
- Many people may argue whether client-server is more general or layered style is more general

Layered style - examples

Java applications

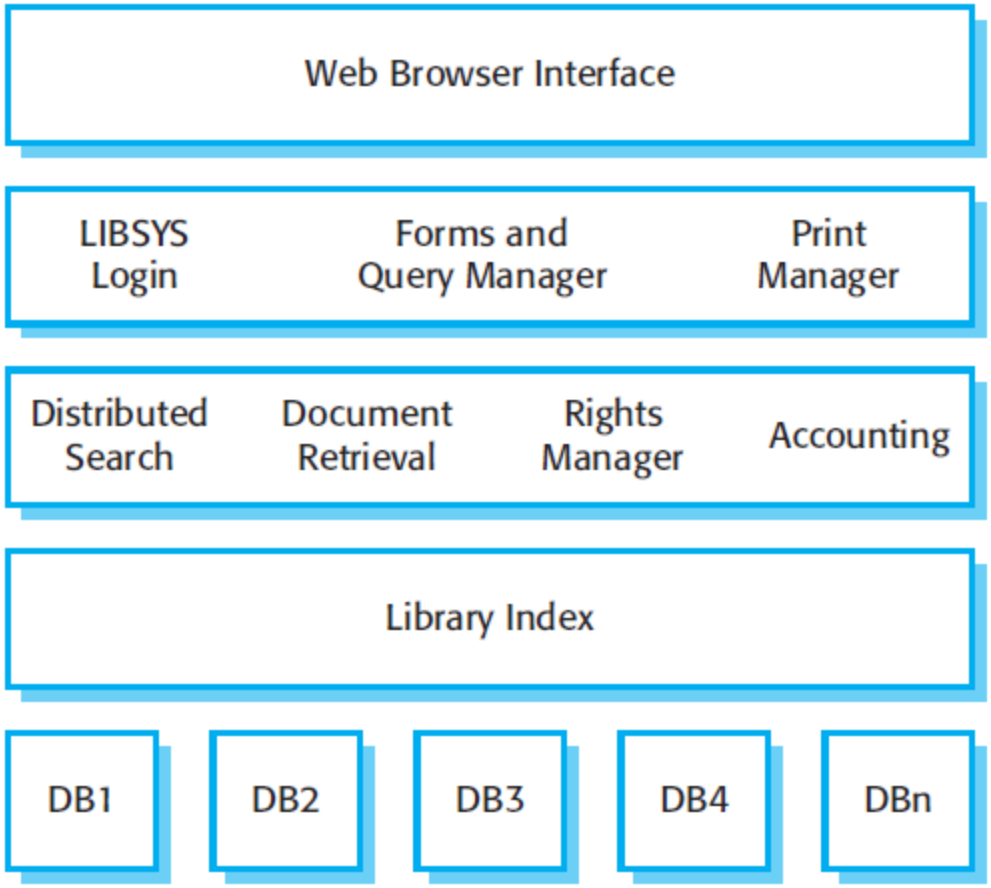


OSI Networking model



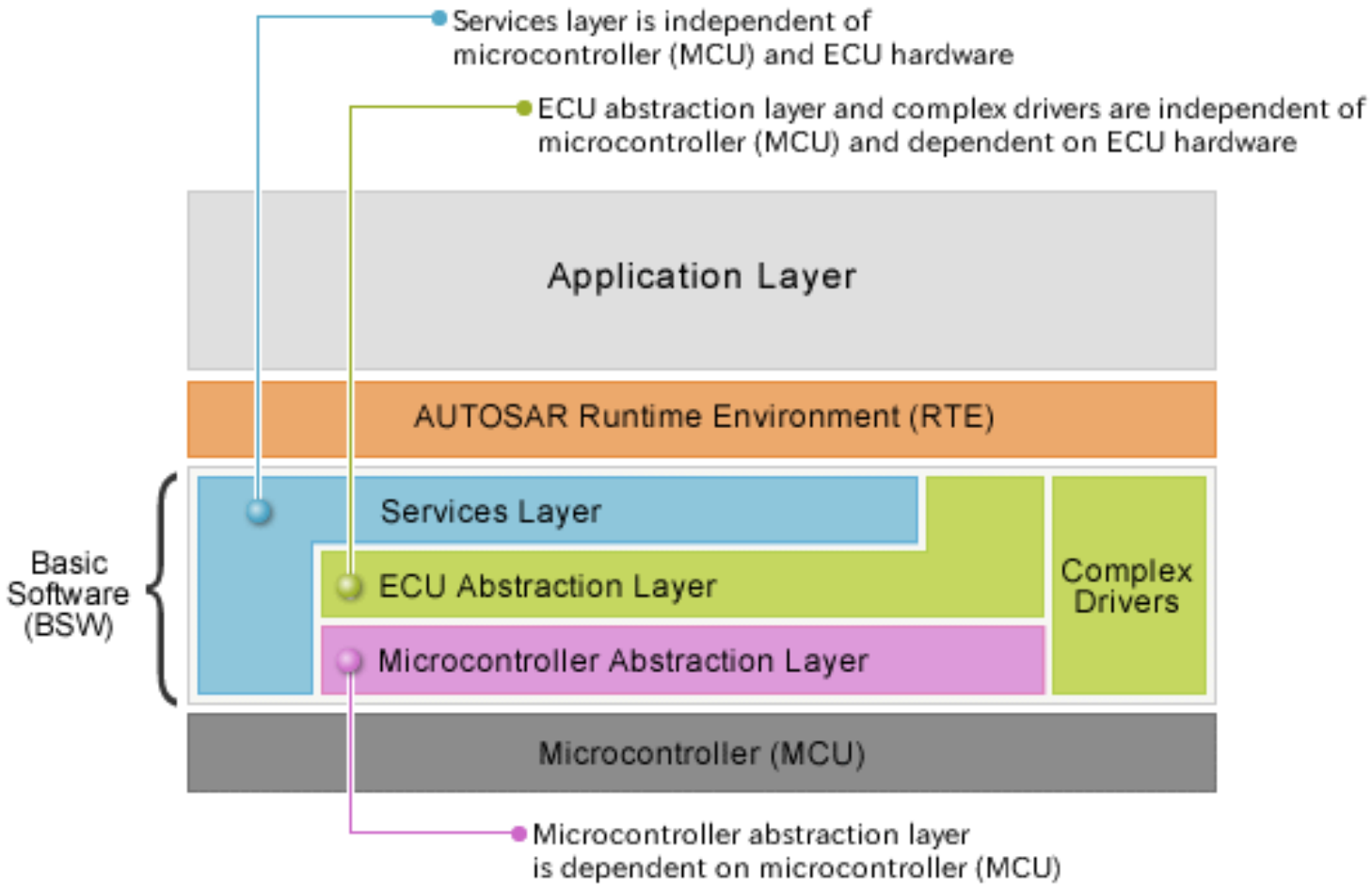
Source:
http://www.windowsnetworking.com/articles-tutorials/common/OSI-Reference-Model-Layer1-hardware.html

Library information system



Source: Software Engineering by Ian Sommerville, 9th edition (2010), Addison-Wesley Pub Co;

Vertical layers



Source: AUTOSAR Layered Architecture
http://www.renesas.eu/applications/automotive/technology/autosar/peer/autosar_layerdarch.jsp

Advantages of the layered style

- Internal structure of the layers is hidden, if the interface is supported
- Abstraction – minimize complexity
- Better cohesion – each layer maintains similar tasks
- Introducing “stub” layers may improve testing

Disadvantages of layered style

- For many systems it is difficult to distinguish separate layers and this lead to increase in design efforts
- Strict layer communication restrictions compromise performance
  - Sometimes *vertical* layers may be implemented

Object-Oriented Style

- Objects represent computational units
  - Objects are responsible for their internal representation integrity
  - Internal representation is hidden from other objects
- Connectors are messages and/or method invocations

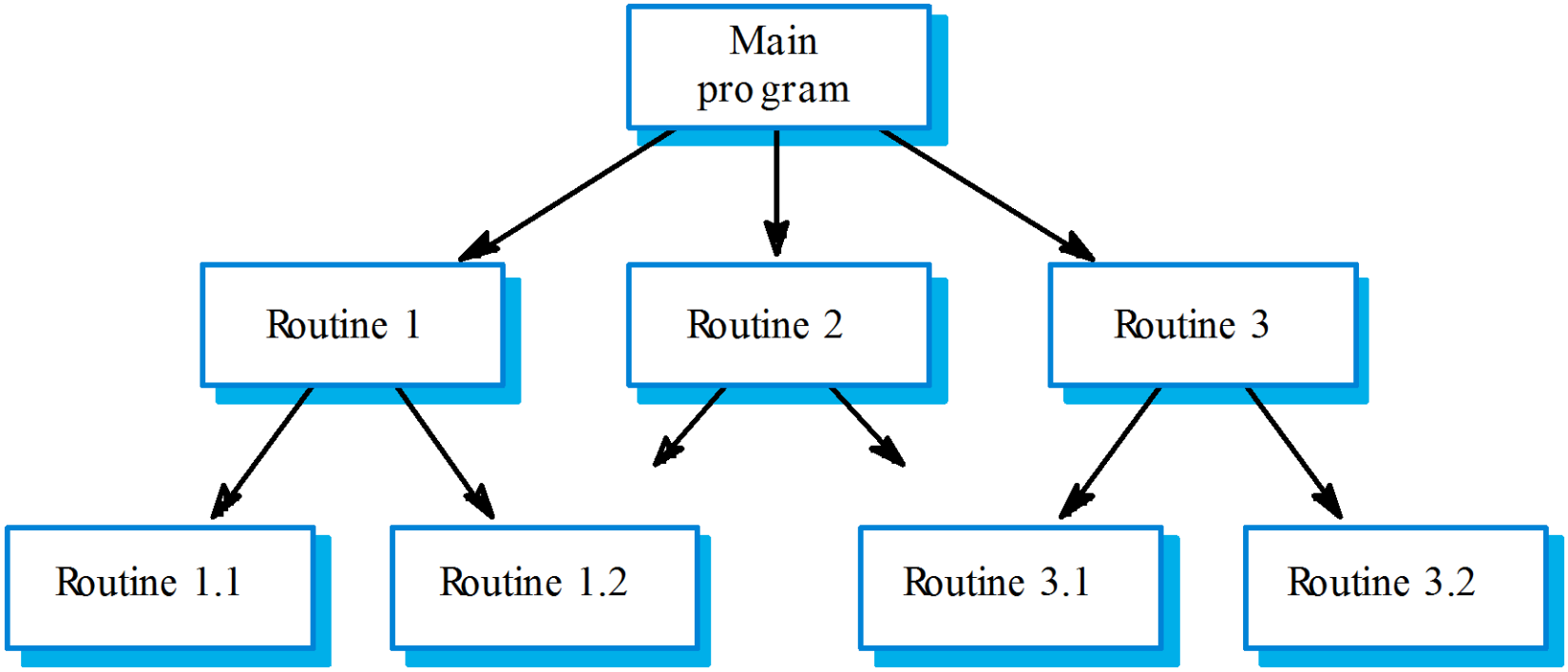
Object-oriented style

- Advantages
  - Encapsulation of data and program logic
  - System decomposition into sets of interacting agents
- Disadvantages
  - Objects must know identities of other objects in order to interact with them
  - Side effects in object method invocations

Implicit invocation style

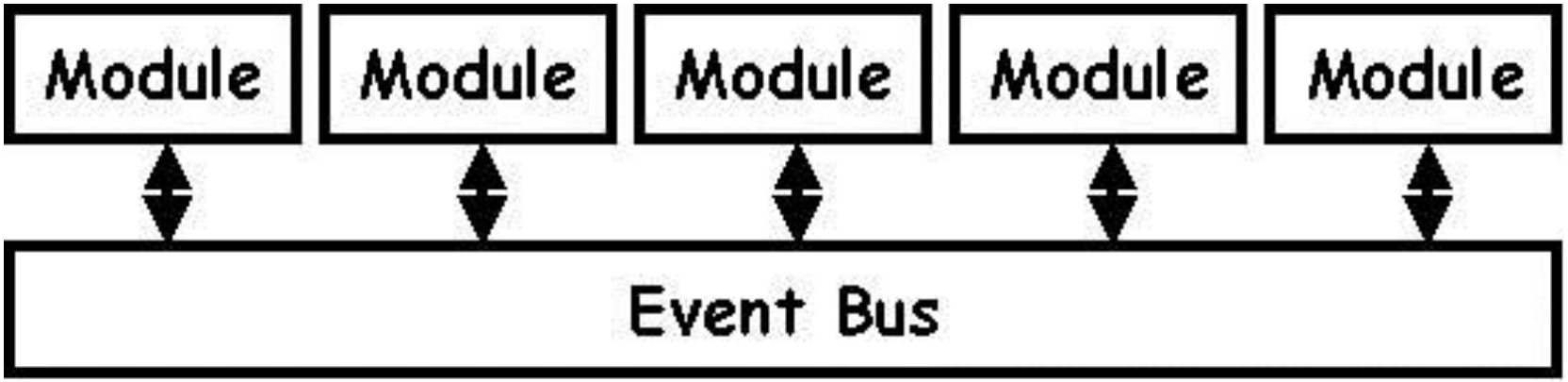
- Components within the system interact with each other by emission of events
- Events may contain not only control messages but also data
- Other names
  - Publish-subscribe
  - Event-based style
  - Message passing style

Explicit invocation

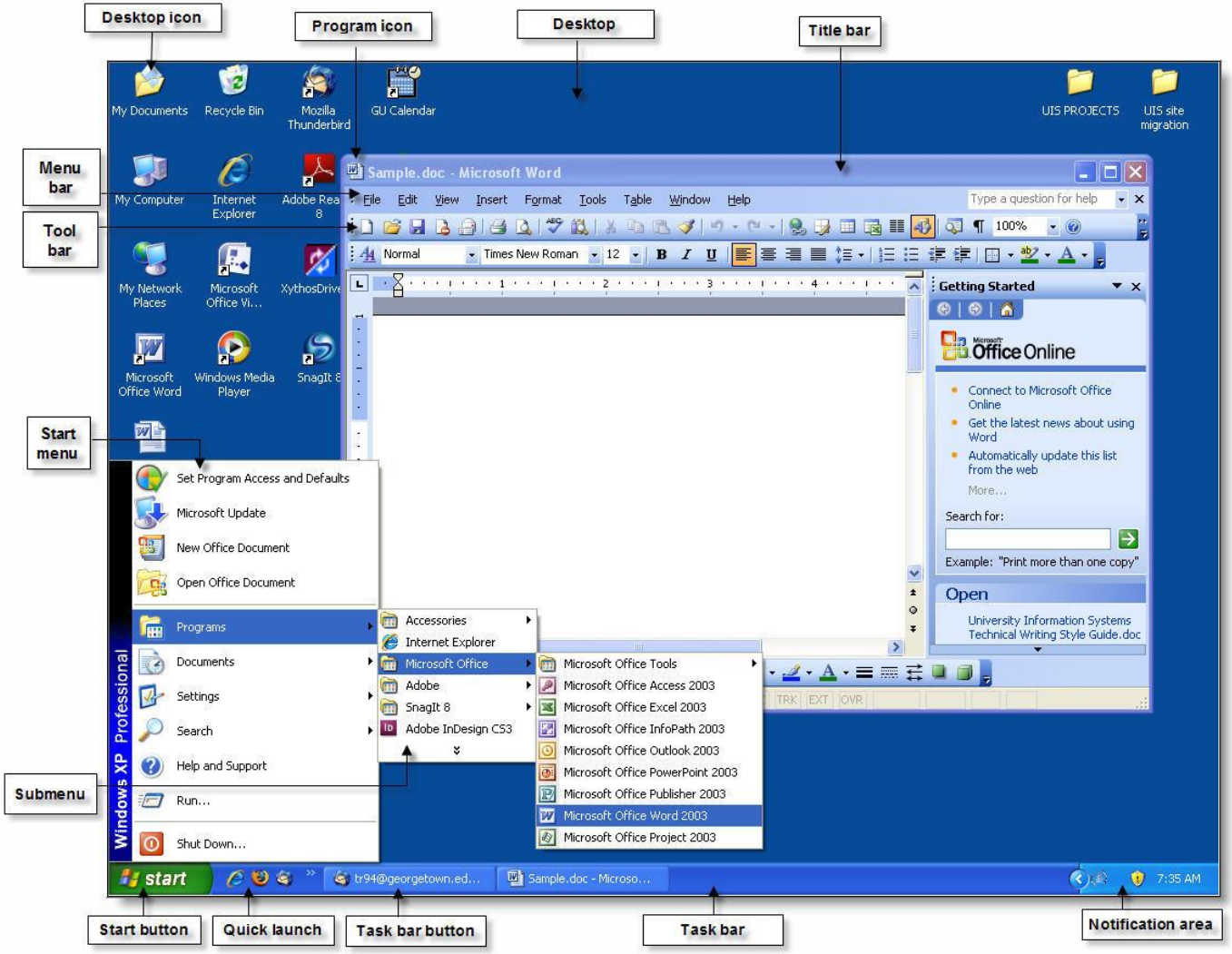


Implicit invocation style

- Components of this style are running concurrently and communicate by receiving or emitting events
- Connector is an event bus
  - All component interact via the bus



Example of implicit invocation style



User interactions are passed to the application as events

Advantages of implicit invocation style

- Louse coupling
  - Components may be very heterogeneous
  - Components are easy to replace or reuse
- Big effectiveness for distributed systems - events are independent and can travel across the network
- Security – events are easily tracked and logged

Disadvantages of implicit invocation style

- Vague structure of the system
  - Sequence of component executions is difficult to control
  - Hard debugging
- It is not sure if there exist a component to react to a given event
- Big amounts of data are difficult to be carried by events
- Reliability issue – malfunction of the event bus will bring the whole system down